Low-Complexity Array Codes for Random and Clustered 4-Erasures

Yuval Cassuto, *Member, IEEE,* and Jehoshua Bruck, *Fellow, IEEE*

***Abstract*—A new family of low-complexity array codes is proposed for correcting 4 column erasures. The new codes are tailored for the new error model of clustered column erasures that captures the properties of high-order failure combinations in storage arrays. The model of clustered column erasures considers the number of erased columns, together with the number of clusters into which they fall, without pre-defining the sizes of the clusters. This model addresses the problem of correlated device failures in storage arrays, whereby each failure event may affect multiple devices in a single cluster. The new codes correct essentially all combinations of clustered 4 erasures, i.e. those combinations that fall into three or less clusters. The new codes are significantly more efficient, in all relevant complexity measures, than the best known 4-erasure correcting codes. These measures include encoding complexity, decoding complexity and update complexity.**

***Index Terms*—Array codes, clustered erasures, correlated failures, storage arrays**

I. Introduction

ARRAY codes have long become a pivotal tool for protecting data reliability and availability in multi-device storage systems. Initially, only trivial codes were used: the repetition code in RAID-1 arrays, and the parity code in RAID-5 arrays[1]. With the scaling of device capacities, demand had grown for higher failure protection, using redundancy-efficient schemes. Consequently, 2-erasure correcting array codes are being deployed in RAID-6 storage arrays. But even this increased erasure correction capability was shown to be insufficient against failure clustering, which results from correlated failure events [12]. That issue of clustered high-order failures due to rare catastrophic events motivates the construction of array codes that specifically target clustered erasures. Such codes become an attractive option if they are able to alleviate the high implementation complexity of generic high-order erasure-correcting codes. The main implementation bottlenecks of high-order array codes are their encoding and decoding complexities – and more dominantly – their update complexity. A high complexity of updates means that writes to the array are slowed down due to the need to update many parity bits. In addition to its toll on performance, a high update complexity increases device wear and shrinks device lifetimes.

Looking on the clustered-failure problem from a coding-theoretic perspective, combating a failure channel that is not memoryless requires the departure from common constructs like *t erasure-correcting codes* and concepts like the *Hamming distance*. These only consider the number of erasures within a code block, and not their relative locations. To capture the failure-clustering phenomenon in a coding-theoretic setting, a precise model definition of clustered erasures is needed. The clustering model proposed in section II classifies erasure combinations by the number of erased columns, and by the number of *clusters* in which the erased columns fall. The number of clusters captures the number of "independent" failure events, each possibly affecting multiple devices in a single cluster of contiguous devices. An example of this characterization of erasure patterns is given in Figure 1 for patterns with $\rho = 4$ erasures. The present erasure characterization is different (and stronger) from the multiple-burst erasure model [2], as it does not predefine the cluster *sizes*, only their *number*. For example, an erasure model that requires correcting $\rho = 4$ erasures in

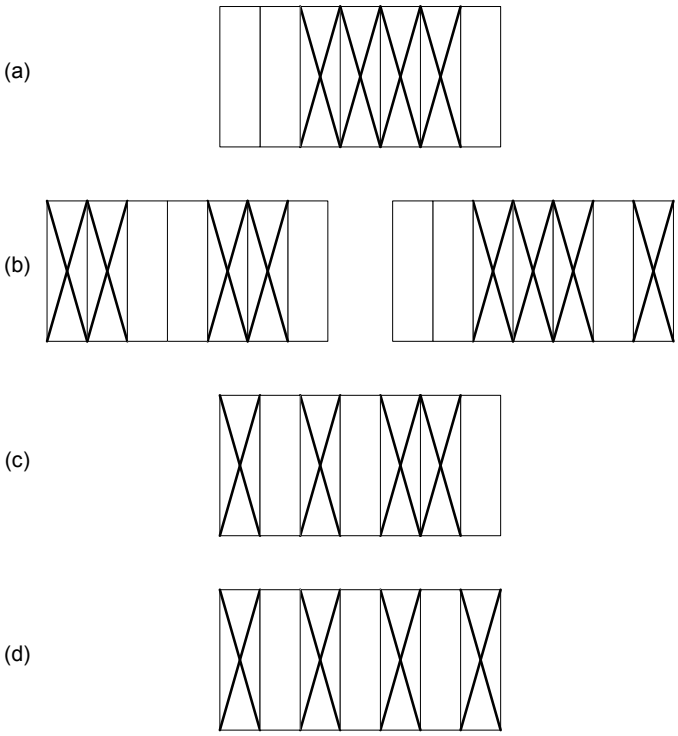

Figure 1. Classification of column combinations by their respective numbers of clusters. Four columns (marked with X) that fall into (a) One cluster (b) Two clusters (c) Three clusters (d) Four clusters (non-clustered)

Yuval Cassuto is with Hitachi Global Storage Technologies, 3403 Yerba Buena Rd., San Jose, CA 95135, U.S.A. (e-mail: yuval.cassuto@hitachigst.com).

Jehoshua Bruck is with the Department of Electrical Engineering, California Institute of Technology, 1200 E California Blvd., Mail Code 136-93, Pasadena, CA 91125, U.S.A. (e-mail: bruck@paradise.caltech.edu).

This work was supported in part by the Caltech Lee Center for Advanced Networking.

[1]The acronym RAID stands for Redundant Array of Inexpensive Disks [13]

up to 2 clusters covers the single erasure burst in (a), and the *two possible patterns* of two clusters in (b) of Figure 1.

This problem of clustered erasures is addressed in this paper for the case of 4 erasures. The main contribution is a strongly-systematic array-code family that corrects all 4-erasures in up to two clusters, and all but a vanishing number of 4-erasures falling into three clusters. The new code family enjoys a significant reduction in complexity compared to the best known 4-erasure array codes. The encoding and decoding complexities are each reduced by 25%, and the update complexity is slashed by 28.57%. That improvement in implementation efficiency comes at the price of degraded *random*-erasure correctability, although 7/8 of the totality of 4-erasure combinations are still correctable.

There have been prior attempts to improve the implementation efficiency of array codes by departing from the requirement to correct all possible 4-erasures [7], [11]. However, these attempts used "black-box" compositions of known codes, which result in poor clustered and random erasure correctability, compared to the results of this paper. The unique property achieved by the new code construction is that it can correct erasure combinations in *both* 2-even + 2-odd columns, and 3-even + 1-odd columns (and the complement 3-odd + 1-even). The 2+2 case alone can be achieved by a standard interleaving of 2-erasure codes, and the 3+1 case alone is corrected by splitting a parity group of a 3-erasure code [7]. To get clustered-erasure correctability, both the 2+2 and 3+1 combinations are required, hence lies the novelty of this paper.

Various aspects of the new code family are studied in the paper. In section IV, the code construction is specified in both geometric and algebraic forms. With clarity in mind, the construction method is presented in two steps: first (sub-section IV-A) the code is specified in a structured form, and then (sub-section IV-B) the column placement is permuted to obtain the final code that enjoys better clustered-erasure correction capability. The code's erasure correction capability is proved in section V. Efficient decoding is described in section VI, and the reliability of arrays that employ the new code is analyzed in section VII, using a Markov probability model. A summarizing comparison between the new code and the best known 4-erasure array code concludes the presentation in section VIII.

II. Definitions and Notations

A. Array codes

The definitions in this sub-section are standard in the area of array codes. The next sub-section presents new notation, specifically for clustered-erasure correction. A *length* n array code consists of n columns. When array codes are used in storage arrays, a column is a model for a physical device or another physical unit of data. In the codes discussed here, there are k columns that store uncoded information bits and r columns that store redundant parity bits (thus $n = k + r$). This array structure has the advantage that information can be read off a device directly without decoding, unless it suffered a failure, in which case a decoding process is invoked. An array code that admits this structure is called *strongly systematic*. A *column erasure* occurs when, for some physical reason, the contents of a particular column cannot be used by the decoder. An erasure is a model for a device failure whereby all the data on the device (or other physical unit) is known to have become unusable. We say that an array with given column erasures is *correctable* by the array code if there exists a decoding algorithm that, independent of the specific array contents, can reconstruct the original array from unerased columns only. An array code is called MDS (*Maximum Distance Separable*) if it has r redundant columns and it can correct all possible combinations of r column erasures. MDS codes obviously have the strongest conceivable erasure correction capability for a given redundancy, since the k information columns can be recovered from *any* k columns. Beyond space efficiency of the code, one should also consider its I/O efficiency. I/O efficiency of a storage array is determined by the *small-write* and *full-column* update complexities of the array code used. The small-write update complexity (often simply called update complexity) is defined as the number of parity-bit updates required for a single information bit update, averaged over all information bits. Appendix A shows how the small-write update complexity is calculated for a sample array code. The full-column update complexity is the number of parity columns that have to be modified per a single full-column update. Another crucial performance measure of an array code is its *erasure-decoding complexity*, defined as the number of bit operations (additions, shifts) required to recover the erased columns from the surviving ones.

B. Random/Clustered erasure correction

To describe column-erasure combinations whose only restriction is the number of erased columns, it is customary to use the somewhat misleading term *random* [9] erasures.

Definition 1. *An array is said to recover from ρ* ***random erasures*** *if it can correct all combinations of ρ erased columns.*

The random erasure model is most natural when storage nodes are known to, or more commonly, assumed to behave uniformly and independent of each other. Indeed, almost all array-code constructions discussed in the literature aim at correcting random erasures. Refinement of the erasure model is possible by adding restrictions on the relative locations of the erased columns. This paper considers *clustered* erasures, where the ρ erasures fall into a limited ($< \rho$) number of clusters. We now turn to some definitions related to the clustered-erasure model. In words, a *cluster* is a contiguous block of columns. More precisely,

Definition 2. *In an array code with columns numbered $\{0, 1, 2, \ldots, n-1\}$, a* ***cluster*** *is a set of σ columns such that the difference between the highest numbered column and the lowest numbered one is exactly $\sigma - 1$.*

For example, $\{2, 3, 4, 5\}$ is a cluster with $\sigma = 4$. Now given a set of columns, the number of clusters that it occupies is the partition of that set to a minimal number of subsets, each of which is a cluster according to the definition above. Now we include another definition that will be useful later.

Definition 3. *A set of ρ columns is called **clustered** if the number of clusters it occupies is* strictly *less than ρ.*

Random erasures have no restriction on their respective numbers of clusters and therefore they include both clustered and non-clustered erasures. The other extreme is the *column burst* model, where all erased columns need to fall into a single cluster. These two well-studied extreme cases open our presentation, and later the new codes are shown to be very effective for all intermediate cases of clustered erasures.

III. Preliminaries and Relevant Known Results

The purpose of this section is to discuss relevant known results in sufficient detail to prepare for the presentation of the new code family in the next section. It also presents the mathematical framework that is used to prove the new code's correction properties.

A. Codes for erasures in a single cluster

Assume that our design goal is an array that will sustain any erasure of ρ columns in a single cluster, without requiring any random-erasure correction capability. A simple and well known technique called *interleaving* can achieve that task optimally both with respect to the required redundancy and in terms of the code update complexity.

Let $\mathcal{CP}$ be an array code with n' columns, out of which $k' = n' - 1$ are information columns. The remaining column holds the bit-wise parity of the k' information columns. Define the code $\mathcal{CP}_\rho$ as the length $n = \rho n'$ code that is obtained by the interleaving of ρ codewords of $\mathcal{CP}$. In other words, if $\mathsf{C}^{(1)}, \mathsf{C}^{(2)}, \ldots, \mathsf{C}^{(\rho)}$ are ρ codewords of $\mathcal{CP}$, then the corresponding code word of $\mathcal{CP}_\rho$ will be

$\mathsf{C}_1^{(1)}$	$\cdots$	$\mathsf{C}_1^{(\rho)}$	$\mathsf{C}_2^{(1)}$	$\cdots$	$\mathsf{C}_2^{(\rho)}$	$\mathsf{C}_3^{(1)}$	$\cdots$

Proposition 4. *The code $\mathcal{CP}_\rho$ corrects any ρ erasures in a single cluster.*

Proof: Any erasure that is confined to at most ρ consecutive columns erases at most one column of each constituent $\mathcal{CP}$ code. These single erasures are correctable by the individual $\mathcal{CP}$ codes. ∎

It is clear that the code $\mathcal{CP}_\rho$ has optimal redundancy since it satisfies $\rho = r$ and ρ is a well known and obvious lower bound on the redundancy r. For any ρ, the code $\mathcal{CP}_\rho$ has update complexity (both small-write and full-column) of 1, which is optimal since a lower update complexity would imply at least one code bit that is independent of all other bits, and erasure of that bit would not be correctable.

B. Codes for random erasures: EVENODD

At the other extreme of the erasure-clustering classification are codes that correct any ρ random erasures. For the special case of $\rho = 4$, which is the case addressed in the current paper, the best known random-erasure correcting codes, in terms of implementation complexity, is the family of generalized EVENODD codes [4]. The generalized ($r = 4$) EVENODD codes are defined over arrays with dimensions $(p-1) \times (p+4)$, with p information columns and 4 parity columns. Discussing generalized EVENODD in depth is beyond the scope of this paper, so we only mention their key properties that are relevant to the current presentation.

- When p is a prime such that 2 is a primitive element in the Galois Field GF(p), they correct any random 4-erasure (hence they are MDS codes for these parameters).
- Their asymptotic small-write update-complexity is $7 - o(1)$. $o(1)$ refers to terms that tend to zero as the code length goes to infinity. Their full-column update-complexity is 4.
- The best known way to decode them is using the algorithm of [5] over the polynomial ring $\mathcal{R}_p$ (to be defined later), for which the decoding complexity is dominated by the term $4kp$.

As the state-of-the-art in correcting 4-erasures, the generalized ($r = 4$) EVENODD codes are used as comparison to the new codes constructed herein.

C. Mathematical framework

We now describe the mathematical framework, borrowed from Blaum-Roth [5], to present the new codes. The length $p-1$ columns of the code array are viewed as polynomials of degree $\leqslant p-2$ over the finite field $\mathbb{F}_2$, taken modulo the polynomial $M_p(x)$, where $M_p(x) = (x^p+1)/(x+1) = x^{p-1} + x^{p-2} + \cdots + x + 1$ (recall that in $\mathbb{F}_2$ summation and subtraction are the same and both done using the Boolean eXclusive OR function). According to that view, the polynomial for a binary column vector $\boldsymbol{c} = [c_0, \ldots, c_{p-2}]^T$ is denoted $c(\alpha) = c_0 + c_1\alpha + \cdots + c_{p-2}\alpha^{p-2}$. Bit-wise addition modulo 2 of two columns is equivalent to summing the corresponding polynomials in the ring of polynomials modulo $M_p(x)$, denoted $\mathcal{R}_p$. Multiplying $c(\alpha)$ by α results in a downward shift of $\boldsymbol{c}$ if c_{p-2} is zero. In the case $c_{p-2} = 1$, multiplying by α requires reduction modulo $M_p(x)$, and thus $\alpha c(\alpha)$ is obtained by first downward shifting $[c_0, \ldots, c_{p-3}, 0]^T$, and then inverting all the bits of the shifted vector. The ring $\mathcal{R}_p$ allows an algebraic representation of codes whose encoding rules comprise column bit-wise additions (ring addition) and column shift-and-invert operations (ring multiplication).

Throughout the paper, we assume that the prime number p is chosen such that 2 is a primitive element in GF(p), hence $M_p(x)$ is irreducible, and the ring $\mathcal{R}_p$ becomes a *finite field* [10, p.197]. The correctability of erasure patterns is proved by showing that the determinant of sub-matrices of the code parity-check matrix are non-zero in the field $\mathcal{R}_p$ (note that the field $\mathcal{R}_p$ is really GF(2^{p-1}), with a specific mapping from length $p-1$ vectors to field elements).

IV. Code Construction

Referring to Figure 2, the proposed code family has $2p$ information columns (white) of $p-1$ bits each, and 4 parity columns (shaded) with the same number of bits. With clarity in mind, we present the new code family in two steps. The first step, included in sub-section IV-A, orders the information columns in a way that reveals their structure. Then, in sub-section IV-B, the order of the information columns is permuted to achieve better clustered-erasure correction.

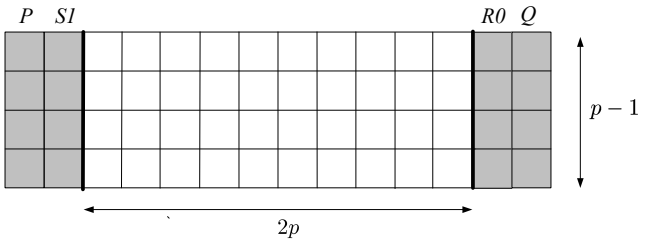

Figure 2. The proposed code array with $2p$ information columns and 4 parity columns. The column size is $p-1$.

A. The structured definition of the code

The information columns are numbered in ascending order from left to right using the integers $\{0,1,2,\ldots,2p-1\}$. Parity columns are not numbered and we use letter labels for them: $\{P, S1, R0, Q\}$. Each of the $p-1$ bits in a parity column is computed from the bit content of its parity group. The structure of the parity groups is now explained via a graphical illustration, for the example of $p=5$. For each of the four parity columns depicted in the four arrays in Figure 3, array locations with the same shape indicate that they belong to the same parity group. Similarly to the EVENODD code [3], parity groups are constrained by the code to have either even or odd parity, depending on the instantaneous array contents, as will be specified shortly. Parity column P, located at the left most column, is simply the bit-wise even parity of the $2p$ information columns. Parity column $S1$, located second from left, is the slope -1 diagonal parity of the *odd* numbered information columns $\{1,3,\ldots,2p-1\}$. The bit groups of $S1$ are set to have even parity if the bits marked EO have even parity, and odd parity otherwise. Parity column Q, located at the right most column, is the XOR of the slope 1 diagonal parities of both the even numbered columns and the odd numbered columns. Parity column $R0$, located second from right, is the slope 2 diagonal parity of the *even* numbered information columns $\{0,2,\ldots,2p-2\}$. The parity groups of Q and $R0$, similarly to those of $S1$, are set to be even/odd, based on the parity of the corresponding EO groups. Note that parity columns P and Q can be decomposed into $P = P0 \oplus P1$ and $Q = Q0 \oplus Q1$, respectively, where $P0, Q0$ depend only on even information columns and $P1, Q1$ only on odd ones. An important fact that will be used in subsequent sections is that even information columns with the parity bits of $P0, Q0, R0$ constitute an $r=3$ MDS code [4], and odd information columns with the parity bits of $P1, Q1, S1$ constitute a (different) $r=3$ MDS code [8].

For a formal definition of the code we include the explicit encoding functions. Denote by $c_{i,t}$ the bit in row i of information column t. For an integer l, define $\langle l \rangle$ to be $l \bmod p$. The formulas to compute the i^{th} bit of each of the parity columns $P, Q, R0, S1$ are provided in Figure 4.

The encoding of information bits into a code array is illustrated in the example of Figure 5.

Algebraic description

An equivalent description of the parity constraints of Figure 4 is the code's parity-check matrix over $\mathcal{R}_p$, which appears in the following for the case $p=5$.

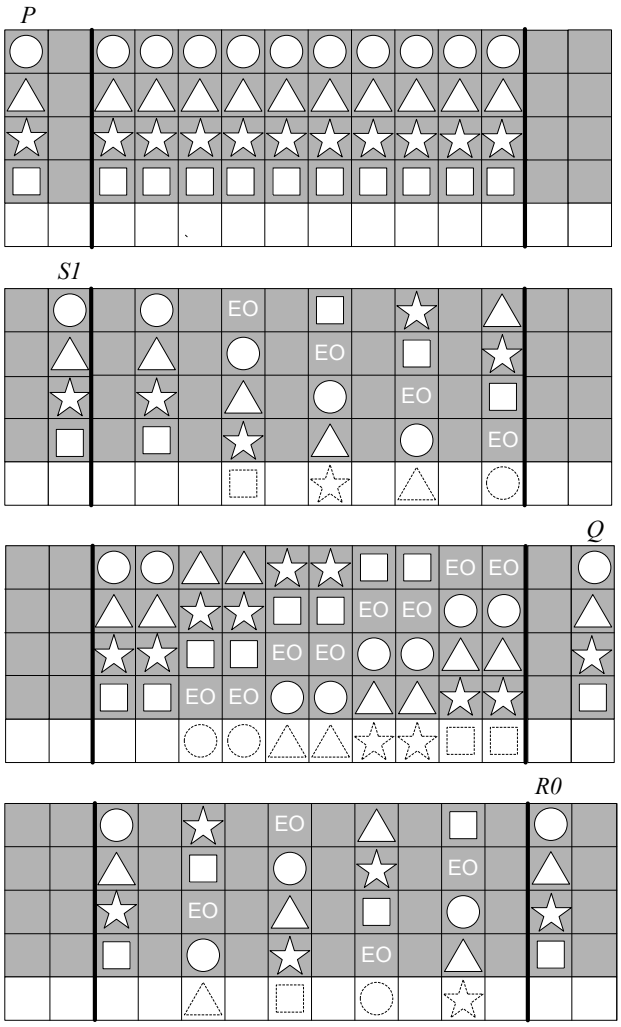

Figure 3. Parity groups for $p=5$. From top to bottom: the parity groups of parity column P (slope 0), $S1$ (slope -1), Q (slope 1) and $R0$ (slope 2). Parity columns $R0$ and $S1$ each depend on only half of the columns, contributing to the low implementation complexity of the code.

$$\left[\begin{array}{cc|cccccccccc|cc} 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & \alpha^4 & 0 & \alpha^3 & 0 & \alpha^2 & 0 & \alpha & 0 & 0 \\ 0 & 0 & 1 & 0 & \alpha^2 & 0 & \alpha^4 & 0 & \alpha & 0 & \alpha^3 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & \alpha & \alpha & \alpha^2 & \alpha^2 & \alpha^3 & \alpha^3 & \alpha^4 & \alpha^4 & 0 & 1 \end{array}\right]$$

The correspondence between the parity groups depicted in Figure 3 and the parity-check matrix above is as follows. The columns of the parity-check matrix correspond to columns of the code array. The two left most columns are for parity columns P and $S1$, and the two right most columns are for $R0$ and Q. Columns in between correspond to information columns in the array. In the parity-check matrix, row 1 represents the constraints enforced by parity column P, rows $2,3,4$ similarly represent the parity constraints of $S1, R0, Q$, respectively. In any row i, the difference of exponents of α in two different columns is exactly the relative vertical shift of the two columns in the shape layout of the appropriate parity in Figure 3. For example, in the top row, all information columns have the same element, $1(=\alpha^0)$, to account for the identical

$$P_i = \bigoplus_{j=0}^{2p-1} c_{i,j}$$

$$S1_i = EO_1 \oplus \bigoplus_{j=0}^{p-1} c_{\langle i+j\rangle,2j+1}\ ,$$

$$\text{where}\quad EO_1 = \bigoplus_{j=0}^{p-1} c_{\langle p-1+j\rangle,2j+1}$$

$$R0_i = EO_0 \oplus \bigoplus_{j=0}^{p-1} c_{\langle i-2j\rangle,2j}\ ,$$

$$\text{where}\quad EO_0 = \bigoplus_{j=0}^{p-1} c_{\langle p-1-2j\rangle,2j}$$

$$Q_i = EO_Q \oplus (\bigoplus_{j=0}^{p-1} c_{\langle i-j\rangle,2j}) \oplus (\bigoplus_{j=0}^{p-1} c_{\langle i-j\rangle,2j+1})\ ,$$

$$\text{where}\quad EO_Q = (\bigoplus_{j=0}^{p-1} c_{\langle p-1-j\rangle,2j}) \oplus (\bigoplus_{j=0}^{p-1} c_{\langle p-1-j\rangle,2j+1})$$

Figure 4. Explicit specification of parity constraints.

vertical alignment of the shapes in the encoding rule of parity P. For general p the parity check matrix H has the following form.

$$H = \left[\begin{array}{cc|cccccccc|cc} 1 & 0 & 1 & 1 & \cdots & 1 & 1 & 1 & \cdots & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & \cdots & 0 & \alpha^{-j} & 0 & \alpha^{-(j+1)} & \cdots & \alpha & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & \alpha^{2j} & 0 & \alpha^{2(j+1)} & 0 & \cdots & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & \cdots & \alpha^{j} & \alpha^{j} & \alpha^{j+1} & \alpha^{j+1} & \cdots & \alpha^{p-1} & 0 & 1 \end{array}\right] \quad (1)$$

B. The permuted definition of the code

In the previous sub-section, the layout of information columns with respect to the parity groups was done in an order that reveals the code's structure. Nevertheless, it turns out that using this structured order for the actual layout of columns in the array does not provide the optimal clustered-erasure correctability. In this sub-section we specify a mapping from the information-column numbers in the structured definition to information column numbers in the actual code specification, which is later called the *permuted code*. This mapping is given as a permutation ψ on the set $\{0,\ldots,2p-1\}$.

$$\psi(t) = (2t) \bmod (2p) + t \bmod 2,\ t \in \{0,\ldots,2p-1\}.$$

Proposition 5. *ψ is a permutation on $\{0,\ldots,2p-1\}$.*

Proof: It is first observed that $0 \leqslant \psi(t) < 2p$, because there is no index t with $(2t) \bmod (2p) = 2p-1$. Now if ψ is not a permutation, then there exist distinct indices t, l such that $(2t) \bmod (2p) + t \bmod 2 = (2l) \bmod (2p) + l \bmod 2$.
Case 1: Both t, l are odd or both are even, and assume without loss of generality $t < l$. Then $(2t) \bmod (2p) = (2l) \bmod (2p)$. And consequently $2l = 2t + 2p$, and in turn $l = t + p$. Since p is odd, the last equation is a contradiction to the assumption that both have the same parity.
Case 2: Without loss of generality t is even and l is odd. Then $(2t) \bmod (2p) = (2l) \bmod (2p) + 1$. This is a contradiction since the left hand side is even while the right hand side is odd. ■

The permutation ψ has the following important properties.

Property 1: $\psi(t) \equiv t \pmod 2$ (odd indices are mapped to odd indices and even indices are mapped to even indices).

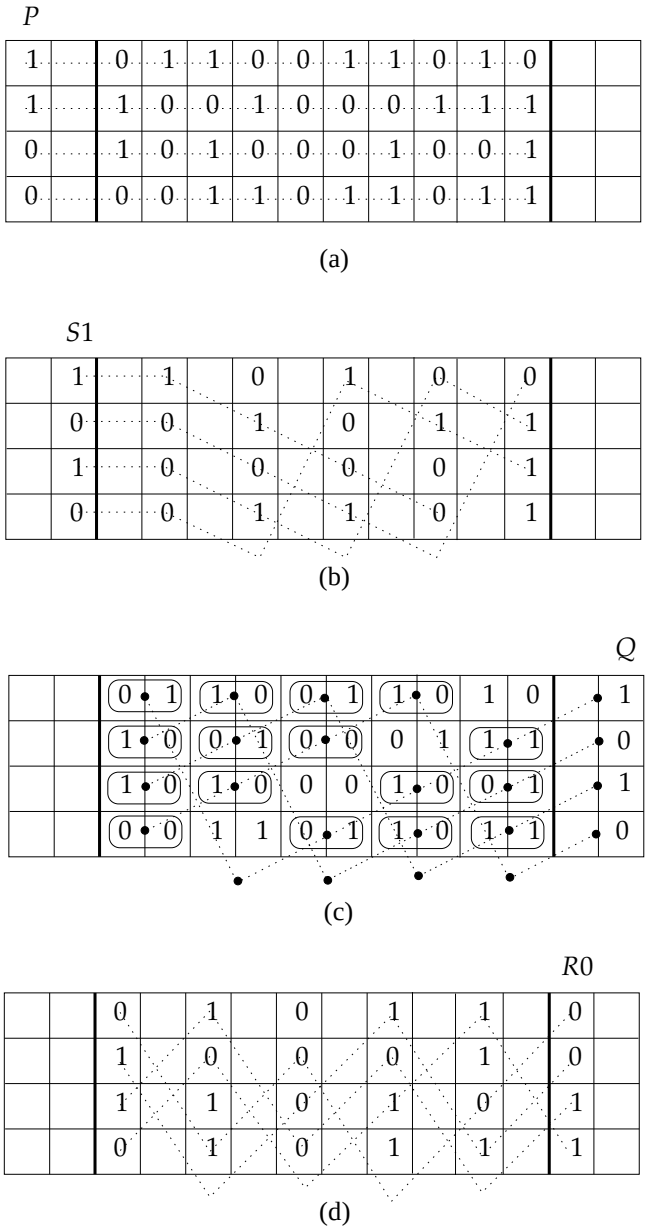

Figure 5. Encoding example. Each parity group from Figure 3 is shown here traversed by a dotted line. (a) Parity column P (always even parity) (b) The groups of parity column $S1$ have odd parity since the non-traversed (EO) bit group has an odd number of ones. (c) The groups of parity column Q have even parity since the non-traversed bit group (EO) has an even number of ones. (d) The groups of parity column $R0$ have odd parity since the non-traversed bit group (EO) has an odd number of ones.

Property 2: A pair of indices $2j$ and $2j+p$ are mapped to a pair of adjacent indices $2l$ and $2l \pm 1$.

The inverse permutation ψ^{-1} is defined next.

Proposition 6. *The inverse permutation of ψ is*

$$\psi^{-1}(s) = \left\lfloor \frac{s}{2} \right\rfloor + p \left(\left\lceil \frac{s \bmod 4}{2} \right\rceil \bmod 2 \right)$$

Proof: We first observe that for $s \in \{0, \ldots, 2p-1\}$, the expression for $\psi^{-1}(s)$ satisfies $0 \leqslant \psi^{-1}(s) < 2p$, since $\lfloor s/2 \rfloor < p$. Now we write the expression for $\psi(\psi^{-1}(s))$.

$$\begin{aligned} \psi(\psi^{-1}(s)) &= (2\psi^{-1}(s)) \bmod (2p) + \psi^{-1}(s) \bmod 2 \\ &= 2\left\lfloor \frac{s}{2} \right\rfloor + \psi^{-1}(s) \bmod 2 \qquad (2) \\ &= s - s \bmod 2 + \psi^{-1}(s) \bmod 2 \\ &= s - s \bmod 2 + s \bmod 2 \qquad (3) \\ &= s \end{aligned}$$

(2) is because $2p \cdot x \equiv 0 \pmod{2p}$ for any x. (3) is from the fact that $\psi^{-1}(s) \equiv s \pmod 2$, which can be verified by evaluating $\psi^{-1}(s)$ for all four different modulo-4 values of s. ∎

The permuted code is now defined using ψ: at column t of the permuted code the column $\psi(t)$ of the structured code is placed (said another way, column s of the structured code is placed at column $\psi^{-1}(s)$ of the permuted code). This results in the parity-check matrix of the permuted code to be

$$H' = \left[\begin{array}{cc|ccccccccc|cc} 1 & 0 & 1 & 1 & 1 & \cdots & 1 & 1 & 1 & 1 & \cdots & 1 & 0 & 0 \\ 0 & 1 & 0 & \alpha^{-1} & 0 & \cdots & 0 & 1 & 0 & \alpha^{-2} & \cdots & \alpha & 0 & 0 \\ 0 & 0 & 1 & 0 & \alpha^4 & \cdots & \alpha^{p-2} & 0 & \alpha^2 & 0 & \cdots & 0 & 1 & 0 \\ 0 & 0 & 1 & \alpha & \alpha^2 & \cdots & \alpha^{p-1} & 1 & \alpha & \alpha^2 & \cdots & \alpha^{p-1} & 0 & 1 \end{array}\right] \qquad (4)$$

From property 2 above, the bottom row has changed from $[1, 1, \alpha, \alpha, \ldots, \alpha^{p-1}, \alpha^{p-1}]$ in H (equation (1)) to $[1, \alpha, \ldots, \alpha^{p-1}, 1, \alpha, \ldots, \alpha^{p-1}]$ in H' (equation (4)). This modified order will be shown in the next section to eliminate uncorrectable erasure combinations with 2 or 3 clusters.

V. Erasure Correctability of the Permuted Codes

In this section we prove that the permuted code specified in the previous section can correct all 4-erasures falling into two or less clusters, as well as all but a vanishing number of 4-erasures falling into three clusters. Hence the proposed code family can correct essentially all clustered erasures. Moreover, considering random-erasure correctability, we prove that a $7/8$ portion of *all* combinations of 4 erasures are correctable by the code.

A. Correction lemmas

We start by a sequence of lemmas that will be used in the next sub-sections to prove the clustered and random erasure correctability of the codes. Recall that the $2p+4$ columns of the code are labeled $\{P, S1, 0, 1, \ldots, 2p-2, 2p-1, R0, Q\}$. Note that Lemma 7 and Lemma 8 are agnostic to whether the structured or the permuted code is used, thanks to the property that even information columns in the structured code are mapped to even information columns in the permuted code, and odd information columns in the structured code are mapped to odd information columns in the permuted code (Property 1 in sub-section IV-B). On the other hand, Lemma 9 specifically refers to the structured code, since it specifies column indices beyond their even/odd property.

The first lemma uses reduction to $r = 3$ codes for 3 even + 1 odd or 3 odd + 1 even erasure combinations.

Lemma 7. *For a combination of 4 erasures, if 3 columns are either even numbered information columns or parity columns in $\{R0, P, Q\}$, and 1 column is an odd numbered information column or the parity column $S1$, then it is a correctable 4-erasure. The complement case: 3 odd (or $S1$ or P or Q) and 1 even (or $R0$), is correctable as well. (in particular, any combination of 3 erasures is correctable).*

Proof: The code can correct the erasure combinations under consideration using a two-step procedure, described for the 3 even + 1 odd case (the complement case is the same, up to changing the identities of parity columns). The first step is to recover the single erased odd information column. Since only one odd column is erased, parity column $S1$ can be used to recover all of its bits. Then, when all odd columns are available, $P1$ and $Q1$ are computed and used to find $P0$ and $Q0$ from P and Q (if not erased) by

$$P0 = P1 \oplus P \quad , \quad Q0 = Q1 \oplus Q$$

After that step, between even information columns and $R0$, $P0$ and $Q0$, only 3 columns are missing. Since even columns, $R0$, $P0$ and $Q0$ constitute an EVENODD code with $r = 3$, the 3 missing columns can be recovered. The complement case of 3 odd and 1 even column erasures is settled by the fact that odd columns with $S1$, $P1$ and $Q1$ constitute an $r = 3$ MDS code [8]. ∎

To get clustered erasure correction, the code should also correct 2 even + 2 odd erasure combinations.

Lemma 8. *When $p > 5$, for a combination of 4 erasures, if 2 columns are even numbered information columns and 2 columns are odd numbered information columns, then it is a correctable 4-erasure.*

Proof: For the case of 2 even and 2 odd information-column erasures we write in (5) the corresponding sub-matrix $\mathcal{M}$ of the parity-check matrix H in a general form.

$$\mathcal{M}^{(j,l,m)} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & \alpha^{-l} & \alpha^{-m} \\ 1 & \alpha^{2j} & 0 & 0 \\ 1 & \alpha^{j} & \alpha^{l} & \alpha^{m} \end{bmatrix} \qquad (5)$$

To prove the correctability of the erasure patterns, the determinant of $\mathcal{M}$ needs to be non-zero for all combinations of j, l, m that satisfy

$$0 \leqslant l < m \leqslant p-1, \quad 0 < j \leqslant p-1$$

Note that assuming that the left most column is column 0 does not limit generality, since any column combination without the 0 column has the same determinant as a combination with the

0 column, up to a multiplication by a non-zero constant in $\mathcal{R}_p$. This fact is proved by observing that shifting the column combination such that one of the even columns is the 0 column is equivalent to multiplying each of rows 2-4 by non-zero constants.

Evaluating the determinant of $\mathcal{M}^{(j,l,m)}$ gives

$$\begin{aligned}\left|\mathcal{M}^{(j,l,m)}\right| &= \alpha^{2j-l+m}+\alpha^{2j+l-m}+\alpha^{-l+m}+\alpha^{l-m} \\ &+ \alpha^{j-m}+\alpha^{2j-m}+\alpha^{j-l}+\alpha^{2j-l} \\ &= (\alpha^j+1)(\alpha^{-l+m}+1)\cdot \\ &\quad \cdot(\alpha^j+\alpha^{j+l-m}+\alpha^{l-m}+\alpha^{j-m}+1)\end{aligned}$$

Since $j>0$ and $m>l$, the first two terms in the product are non-zero. The last term has an odd number of monomials, and therefore cannot be 0. For $p>5$ it also cannot equal $M_p(\alpha)$, the modulus of the field $\mathcal{R}_p$. ■

The next lemma treats additional erasure combinations that include parity columns and that are not covered by Lemma 7.

Lemma 9. *If $p>3$, the following 4-erasure combinations are correctable by the structured code.*

1) *$S1$, 1 odd information column and 2 even information columns*
2) *$R0$, 1 even information column and 2 odd information columns*
3) *$R0$,$S1$, 1 even information column and 1 odd information column, except pairs of information columns numbered $2i, 2i+1$, for $0 \leqslant i \leqslant p-1$.*

Proof: The sub-matrix that corresponds to case 1 is

$$\mathcal{M}_1^{(j,l)} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & \alpha^{-l} \\ 0 & 1 & \alpha^{2j} & 0 \\ 0 & 1 & \alpha^{j} & \alpha^{l} \end{bmatrix}$$

The variables j,l satisfy the conditions

$$0 \leqslant l \leqslant p-1, \quad 0 < j \leqslant p-1$$

Evaluating the determinant of $\mathcal{M}_1$ gives

$$\left|\mathcal{M}_1^{(j,l)}\right| = (\alpha^j+1)(\alpha^{j+l}+\alpha^j+\alpha^l)$$

Since $j>0$, the first term is non-zero. The second term has an odd number of monomials and therefore cannot be 0. For $p>3$ it also cannot equal $M_p(\alpha)$.

The sub-matrix that corresponds to case 2 is

$$\mathcal{M}_2^{(l,m)} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & \alpha^{-l} & \alpha^{-m} & 0 \\ 1 & 0 & 0 & 1 \\ 1 & \alpha^{l} & \alpha^{m} & 0 \end{bmatrix}$$

The variables l,m satisfy the conditions

$$0 \leqslant l < m \leqslant p-1$$

The determinant now equals

$$\left|\mathcal{M}_2^{(l,m)}\right| = (\alpha^{-l}+\alpha^{-m})(\alpha^l+\alpha^m+1)$$

Since $m>l$, the first term is non-zero. The second term has an odd number of monomials and therefore cannot be 0. For $p>3$ it also cannot equal $M_p(\alpha)$.

The sub-matrix that corresponds to case 3 is

$$\mathcal{M}_3^{(l)} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & \alpha^{-l} & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & \alpha^{l} & 0 \end{bmatrix}$$

The variable l satisfies the conditions

$$0 \leqslant l \leqslant p-1$$

The determinant now equals

$$\left|\mathcal{M}_3^{(l)}\right| = \alpha^l+1$$

The determinant above is non-zero if $l>0$, a condition that is equivalent to requiring that the even and odd information columns are not numbered $2i, 2i+1$, respectively, for $0 \leqslant i \leqslant p-1$. ■

B. Clustered erasure correctability

Finally, we are ready to prove the main results of the paper. The structured codes are next shown to correct all 4-erasures in up to two clusters, and asymptotically all 4-erasures in three clusters.

Theorem 10. *For $p>5$, the permuted code corrects all 4-erasures that fall into at most two clusters.*

Proof: If a 4-erasure falls into two or less clusters, then it either has 3 even and 1 odd columns (or the complement) or 2 even and 2 odd columns. If all columns are information columns, the 3+1 case is covered by Lemma 7, and the 2+2 case (for $p>5$) by Lemma 8. If P or Q (or both) are erased, then the remaining columns can be neither all odd (or $S1$) nor all even (or $R0$), cases which are covered by Lemma 7 as well. The case of only one of $S1$ and $R0$ erased is covered by cases 1 and 2 of Lemma 9, respectively. Finally, 4-erasures with both $S1$ and $R0$ erased that are not covered by case 3 of Lemma 9 can *not* fall into two or less clusters. The information columns $2i, 2i+1$ of the structured code are at locations $2l, 2l+p$ in the permuted code. Hence uncorrectable combinations $\{S1, 2i, 2i+1, R0\}$ of the structured code from Lemma 9 map to combinations $\{S1, 2l, 2l+p, R0\}$ that occupy more than two clusters in the permuted code.

■

Theorem 11. *For $p>5$, the ratio between the number of three-cluster 4-erasures correctable by the permuted code and the total number of three-cluster 4-erasures is greater than 0.972. As p goes to infinity, this ratio tends to 1.*

Proof: If a 4-erasure falls into three clusters, then it either has 2 even and 2 odd columns or 3 even and 1 odd columns (or the complement). Lemmas 7, 8 and 9 address all such combinations, except the following. Combinations $\{S1, 0, p, R0\}$ and $\{S1, p-1, 2p-1, R0\}$ mapped from $\{S1, 0, 1, R0\}$ and $\{S1, 2p-2, 2p-1, R0\}$, respectively, which are exceptions

to case 3 of Lemma 9. Also, $\{P, S1, 2i+1, 2j+1\}$ and $\{2i, 2j, R0, Q\}$ cannot be corrected as they are not included in Lemma 7.

Hence the number of non-correctable 4-erasures with three clusters is

$$2+2\binom{p}{2}$$

The total number of 4-erasures with three clusters is

$$3\binom{2p-1}{3}$$

(in general this equals $3\binom{n-3}{3}$ for length n arrays since by taking any choice of 3 points from a set of $n-3$ points on a line, we can uniquely obtain an erasure combination with three clusters, following the procedure below. We first choose 3 points out of the $n-3$ points to be the cluster-start locations. Then the point that represents the cluster with size 2 is selected from these 3 points (for that we have the factor 3). Given these choices, the 3 clusters are obtained by augmenting the size 2 cluster with an additional point to its right and in addition augmenting each of the two left points with a point to its right as a cluster spacer. Hence any such choice gives a 3-cluster.)

Thus the ratio between the number of correctable 3-cluster 4-erasures and the total number of 3-cluster 4-erasures equals

$$\frac{3\binom{2p-1}{3}-2-2\binom{p}{2}}{3\binom{2p-1}{3}} = 1-\frac{p^2-p+2}{4p^3-12p^2+11p-3}$$

For $p=11$ (the smallest prime $p>5$ with 2 primitive in $\mathrm{GF}(p)$), the ratio attains its minimal value of 0.972. Moreover, it is readily seen that this ratio equals $1-o(1)$, where $o(1)$ are terms that tend to zero as p goes to infinity. ■

The only uncorrectable clustered erasure combinations are ones that include parity erasures. Had we lifted the requirement from the codes to be strongly systematic (i.e. to have dedicated columns for parity bits), it would have been possible to correct all clustered erasures. A similar reduction in correction capability of strongly systematic codes was on the part of generalized EVENODD codes [4], compared to the similar non-systematic codes of [5]. While the latter are MDS for any r-erasure correction, the former have r parameters with uncorrectable erasure combinations that include parity columns.

C. Random erasure correctability

The codes are next shown to correct a $7/8$ portion of all combinations of 4 erasures.

Theorem 12. *For $p>5$, the ratio between the number of 4-erasures that are correctable by the permuted code and the total number of 4-erasures is greater than 0.865. As p goes to infinity, this ratio tends to $7/8=0.875$.*

Proof: Building on Lemmas 7, 8 and 9, the number of correctable 4-erasures equals

$$\overbrace{2\binom{p+3}{3}(p+1)-(p+1)^2}^{(1)} + \overbrace{\binom{p}{2}\binom{p}{2}}^{(2)} + \underbrace{2p\binom{p}{2}}_{(3)} + \underbrace{p^2-2}_{(4)}$$

(1), obtained by Lemma 7, is the number of ways to select 3 even information columns (or $R0$ or P or Q) and 1 odd information column (or $S1$), multiplied by 2 to include the complement case, and subtracting the doubly counted combinations with both P and Q.

(2), obtained by Lemma 8, is the number of ways to select 2 even and 2 odd information columns.

(3), obtained by cases 1 and 2 of Lemma 9, is the number of ways to select 2 even information columns and 1 odd information column, multiplied by 2 to include the complement case.

(4), obtained by case 3 of Lemma 9, is the number of ways to select an even information column and an odd information column that are not one of the two combinations $\{S1, 0, p, R0\}$ and $\{S1, p-1, 2p-1, R0\}$, not correctable by Lemma 9.

The total number of 4-erasure combinations is

$$\binom{2p+4}{4}$$

Taking the ratio of the two we obtain

$$\frac{7p^4+34p^3+59p^2+33p+10}{8p^4+40p^3+70p^2+50p+12}$$

For $p=11$, the ratio attains its minimal value of 0.865. Moreover, it is readily seen that this ratio equals $7/8-o(1)$. ■

VI. Efficient Decoding of Erasures

In section V, the decodability of clustered and random erasures was proved by algebraic reasoning. In this section we take a more constructive path and study efficient ways to decode random and clustered erasures. The purpose of this analysis is to prove that decoding of the new code can be done using $3kp+o(p^2)$ bit operations, while the best known algorithm for a 4-erasure correcting MDS code is $4kp+o(p^2)$ [5]. Since k is taken to be in the order of p, saving about kp bit operations gives a quadratic (in p) savings in computations that is very significant in practice for large p.

For the decoding-complexity analysis, we only consider erasure of 4 *information* columns, since these are the most complex cases to decode. We moreover only consider erasures of two even columns and two odd columns, since it was shown in Lemma 7 that the three even and one odd case (or three odd and one even) reduces to decoding three erasures on the even (or odd) columns of the code. Throughout the section we will assume that one of the erased columns is the left-most even information column, as all other cases are cyclically equivalent.

A. Description of 4-erasure decoding algorithm

A general 4-erasure can be decoded using a straightforward procedure over $\mathcal{R}_p$. Ways to perform the steps of that procedure in an efficient way are the topic of the next sub-section. The erased symbols, which represent the content of the erased array columns, are denoted by $\{e_1, o_1, e_2, o_2\}$. e_1, e_2 have even locations and o_1, o_2 have odd locations. First the syndrome vector $\boldsymbol{s}$ of the array is calculated by taking the product

$$\boldsymbol{s} = H\boldsymbol{r}$$

where $\boldsymbol{r}$ is the length $2p+4$ column vector over $\mathcal{R}_p$ that represents the array contents, with erased columns set to the zero element. From the sparsity of the matrix H, the complexity of obtaining the syndrome vector is $3kp$ ($R0$ and $S1$ each checks only half of the columns). Now the erased columns can be directly recovered by

$$\begin{bmatrix} e_1 \\ o_1 \\ e_2 \\ o_2 \end{bmatrix} = E^{-1}\boldsymbol{s} \tag{6}$$

where E denotes the 4×4 sub-matrix of H that corresponds to the 4 erased columns' locations:

$$E = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & \alpha_1^{-1} & 0 & \alpha_3^{-1} \\ 1 & 0 & \alpha_2^2 & 0 \\ 1 & \alpha_1 & \alpha_2 & \alpha_3 \end{bmatrix}.$$

Recall from (1) that each α_i is an element in $\mathcal{R}_p$ of the form α^{l_i}, for some $0 \leqslant l_i \leqslant p-1$. Therefore, E can be written as

$$E = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & \alpha^{-u} & 0 & \alpha^{-w} \\ 1 & 0 & \alpha^{2v} & 0 \\ 1 & \alpha^{u} & \alpha^{v} & \alpha^{w} \end{bmatrix}.$$

The inverse of E, which is used in (6) to decode erasures, is now given in a closed form

$$\begin{aligned} E^{-1} \;=\; & \left(\alpha^u+\alpha^v+\alpha^w+\alpha^{u+v}+\alpha^{v+w}\right)^{-1} \cdot \\ & \begin{bmatrix} 1+\alpha^v & 0 & 0 & 0 \\ 0 & \alpha^u+\alpha^w & 0 & 0 \\ 0 & 0 & 1+\alpha^v & 0 \\ 0 & 0 & 0 & \alpha^u+\alpha^w \end{bmatrix}^{-1} \cdot \\ & \begin{bmatrix} \alpha^{2v}(\alpha^u+\alpha^w) & \alpha^{u+2v+w} & \alpha^u+\alpha^v+\alpha^w & \alpha^{2v} \\ \alpha^{u+v} & \alpha^{u+w}(\alpha^v+\alpha^w+\alpha^{v+w}) & \alpha^u & \alpha^u(1+\alpha^v) \\ \alpha^u+\alpha^w & \alpha^{u+w} & 1+\alpha^u+\alpha^w & 1 \\ \alpha^{v+w} & \alpha^{u+w}(\alpha^u+\alpha^v+\alpha^{u+v}) & \alpha^w & \alpha^w(1+\alpha^v) \end{bmatrix} \end{aligned}$$

From (6) and the closed-form expression above, the erased symbol e_1 can be recovered by the following product

$$\begin{aligned} e_1 = & \left[\left(\alpha^u+\alpha^v+\alpha^w+\alpha^{u+v}+\alpha^{v+w}\right)\cdot\left(1+\alpha^v\right)\right]^{-1} \cdot \\ & \cdot \left[\; \alpha^{2v}(\alpha^u+\alpha^w),\quad \alpha^{u+2v+w},\quad \alpha^u+\alpha^v+\alpha^w,\quad \alpha^{2v} \;\right] \cdot \boldsymbol{s} \end{aligned}$$

Once e_1 is known, e_2 can be recovered using a simple parity completion with the aid of parity column $R0$. The bits of the odd columns are then recovered by a chain of XOR operations with the aid of parity columns P, Q, that can now be adjusted to $P1, Q1$ when all even columns are known.

Calculating e_1 then reduces to the following chain of calculations

1) Finding the inverse of $\left(\alpha^u+\alpha^v+\alpha^w+\alpha^{u+v}+\alpha^{v+w}\right)\left(1+\alpha^v\right)$ over $\mathcal{R}_p$.
2) Multiplication of sparse $\mathcal{R}_p$ elements by dense $\mathcal{R}_p$ elements. The sparse elements are the four elements from the E matrix (that have a small ($\leqslant 3$) constant number of non-zero monomials, for any p) and the dense elements are the four syndrome elements.
3) Multiplication of two dense $\mathcal{R}_p$ elements resulting from the previous steps.

B. Analysis of 4-erasure decoding algorithm

We now analyze the number of bit operations required for each decoding task.

1) **Finding inverses of $\mathcal{R}_p$ elements**:
   The inverse of an element $f(\alpha)\in\mathcal{R}_p$ is the element $\tilde{f}(\alpha)$ that satisfies $\tilde{f}(x)f(x)+a(x)M_p(x) = 1$, for some polynomial $a(x)$. When $f(\alpha)$ is invertible, the polynomial $\tilde{f}(x)$ can be found by the Extended Euclid Algorithm for finding the greatest common divisor of the polynomials $f(x)$ and $M_p(x)$. An efficient algorithm for polynomial greatest common divisors is given in [1, Ch.8] that requires $O(p\log^4 p)$ bit operations ($O(\log p)$ polynomial multiplications, each taking $O(p\log^3 p)$ bit operations, as shown in item 3 below).
2) **Multiplication of a sparse $\mathcal{R}_p$ element by a dense $\mathcal{R}_p$ element** requires $O(p)$ bit operations. Since the number of non-zero monomials in the sparse polynomial is constant in p, the trivial polynomial multiplication algorithm requires $O(p)$ shifts and additions modulo 2.
3) **Multiplication of two dense $\mathcal{R}_p$ elements** can be done in $O(p\log^3 p)$ bit operations using Fourier-domain polynomial multiplication [1, Ch.7]. We describe this procedure for the special case of polynomial coefficients over GF(2). Let $N \geqslant 2p-2$ be the smallest such integer of the form $N = 2^\ell - 1$, where ℓ is an integer. Let ω be a principal Nth root of unity in the finite field GF(2^ℓ). Then ω defines a Discrete Fourier Transform on length N vectors over GF(2^ℓ). The product of two polynomials of degree $p-2$ or less can be obtained by element-wise multiplication of their individual Discrete Fourier Transforms, and then applying the Inverse Discrete Fourier Transform to the resulting length N vector. Using the FFT algorithm, each transformation requires $O(N\log N)$ operations over GF(2^ℓ), or $O(N\log^3 N)$ bit operations. The element-wise multiplication requires N multiplications over GF(2^ℓ), or $O(N\log^2 N)$ bit operations. Since $N < 4p$, the total number of bit operations needed for multiplying two dense $\mathcal{R}_p$ elements is $O(p\log^3 p)$.

Since each of the decoding operations in 1-3 above has a complexity that vanishes with respect to p^2, the $3kp$ complexity of finding the syndrome vector dominates the decoding complexity when k is of the same order as p.

VII. Reliability Analysis of the Code in Storage Arrays

The main motivation for the construction of array codes in general, and the codes of this paper in particular, is to provide efficient protection for storage arrays against device failures. The benefit of deploying an array code in a practical storage system obviously lies in the trade-off it achieves between erasure correction capability and implementation complexity. To this end, the correction capability of the new codes was characterized in detail in section V. The purpose of this section is to project this correction capability onto the *reliability* domain, analyzing the storage array's susceptibility to data loss, using a statistical framework. A common reliability metric for storage arrays is the *expected* time before data loss, denoted $MTTDL$ (Mean Time To Data Loss) [6]. Ultimately, this section will detail a procedure to calculate the $MTTDL$ of storage arrays, when protected by the new codes, in the presence of random and clustered device failures (erasures). This will be done after first presenting the general method of $MTTDL$ calculation as applied in the literature to MDS codes under random erasures.

A. $MTTDL$ calculation for MDS codes under random erasures

Using the method presented in [6, Ch.5] for single-erasure-correcting arrays under random erasures (termed *Independent disk lifetimes* therein), we calculate the $MTTDL$ of all-4-erasure-correcting arrays as an example that is later used for comparison with the new codes. The direct calculation of the $MTTDL$ becomes a simpler task if device failures and repairs follow a Markov process and can thus be described by a Markov state diagram. To allow that, the following assumptions are made.

- Device lifetimes follow an exponential distribution with equal mean[2] $MTTF_{\text{device}} = 1/\lambda$.
- Repair times are also exponential with mean $MTTR_{\text{device}} = 1/\mu$.
- The number of devices is large compared to the number of tolerable failures, so the transition probabilities between states do not depend on the instantaneous number of failed devices.

When those assumptions are met, the reliability of a device array can be described by the state diagram shown in Figure 6. The label of each state represents the number of failed devices.

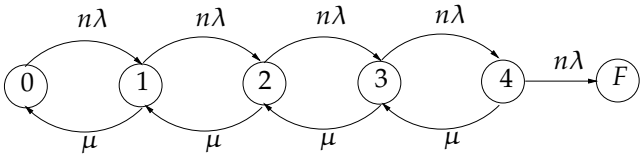

Figure 6. State diagram description of all-4-erasure correcting arrays under random failures. The failure process with rate $n\lambda$ moves to a higher failure state. The repair process with rate μ moves to a lower failure state.

[2]MTTF stands for Mean Time To Failure and MTTR stands for Mean Time To Repair

State F (Fail) represents permanent data loss resulting from a failure count that is above the array's correction capability. The exponential distributions allow specifying the transitions between states in terms of *rates*. The transition rate from lower to higher states is the inverse $MTTF_{\text{device}}$ of individual devices, times the number of devices in the array. The reverse transitions that represent repairs have rates that are the inverse $MTTR_{\text{device}}$ assumed in the system. Using the state diagram, the $MTTDL$ is the expected time to get from state 0 (initial state) to state F (data loss state).

$$MTTDL \triangleq E[0 \to F]$$

The Markov property of the process permits the decomposition

$$E[0 \to F] = E[\text{time stays in } 0] + E[1 \to F] = \frac{1}{n\lambda} + E[1 \to F]$$

Linear relationships between $E[i \to F]$ and $E[j \to F]$ are induced whenever state i and state j are connected. For all-4-erasure correcting arrays (MDS-4), the $MTTDL$ is then obtained as the solution (for $E[0 \to F]$) of the following linear system.

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -\frac{\mu}{\mu+n\lambda} & 1 & -\frac{n\lambda}{\mu+n\lambda} & 0 & 0 \\ 0 & -\frac{\mu}{\mu+n\lambda} & 1 & -\frac{n\lambda}{\mu+n\lambda} & 0 \\ 0 & 0 & -\frac{\mu}{\mu+n\lambda} & 1 & -\frac{n\lambda}{\mu+n\lambda} \\ 0 & 0 & 0 & -\frac{\mu}{\mu+n\lambda} & 1 \end{bmatrix} \begin{bmatrix} E[0\to F] \\ E[1\to F] \\ E[2\to F] \\ E[3\to F] \\ E[4\to F] \end{bmatrix} = \begin{bmatrix} \frac{1}{n\lambda} \\ \frac{1}{\mu+n\lambda} \\ \frac{1}{\mu+n\lambda} \\ \frac{1}{\mu+n\lambda} \\ \frac{1}{\mu+n\lambda} \end{bmatrix}$$

that is found to be

$$MTTDL_{\text{MDS4}} = \frac{1}{\Lambda^5}(5\Lambda^4 + 4\mu\Lambda^3 + 3\mu^2\Lambda^2 + 2\mu^3\Lambda + \mu^4)$$

where $\Lambda \triangleq n\lambda$ is used for notational convenience.

B. $MTTDL$ calculation for the new codes under random and clustered failures

For the model of random failures, the $MTTDL$ of the new codes can be calculated by a straightforward application of the method in the previous sub-section – executed on the transition diagram of Figure 7.

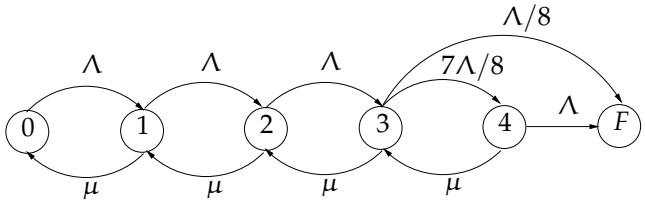

Figure 7. State diagram description of arrays protected with the new codes, under random failures. Since the new codes correct only a $7/8$ ratio of 4-erasures, the failure rate out of state 3 is split to $7\Lambda/8$ into state 4, as before, and $\Lambda/8$ directly into state F.

The corresponding linear system of equations on the 5 active states $0, 1, 2, 3, 4$ is now

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -\frac{\mu}{\mu+\Lambda} & 1 & -\frac{\Lambda}{\mu+\Lambda} & 0 & 0 \\ 0 & -\frac{\mu}{\mu+\Lambda} & 1 & -\frac{\Lambda}{\mu+\Lambda} & 0 \\ 0 & 0 & -\frac{\mu}{\mu+\Lambda} & 1 & -\frac{7}{8}\cdot\frac{\Lambda}{\mu+\Lambda} \\ 0 & 0 & 0 & -\frac{\mu}{\mu+\Lambda} & 1 \end{bmatrix} \begin{bmatrix} E[0\to F] \\ E[1\to F] \\ E[2\to F] \\ E[3\to F] \\ E[4\to F] \end{bmatrix} = \begin{bmatrix} \frac{1}{\Lambda} \\ \frac{1}{\mu+\Lambda} \\ \frac{1}{\mu+\Lambda} \\ \frac{1}{\mu+\Lambda} \\ \frac{1}{\mu+\Lambda} \end{bmatrix}$$

The solution of that system gives

$$MTTDL_{\text{new,rand}} = \frac{39\Lambda^4 + 35\mu\Lambda^3 + 26\mu^2\Lambda^2 + 17\mu^3\Lambda + 8\mu^4}{8\Lambda^5 + \mu\Lambda^4}$$

The exact $MTTDL$ calculations are now used to compare the reliability of the new codes to the reliabilities of all-4-erasure and all-3-erasure correcting codes. For the comparison, Λ is fixed to be $100/8760$[1/hr], which applies e.g. to an array with 100 devices and $MTTF_{\text{device}} = 1$[Year]. The $MTTDL$ in hours ([hr]) is then calculated for repair rates μ between 0.01[1/hr] and 10[1/hr]. The graph in Figure 8 shows that the new codes

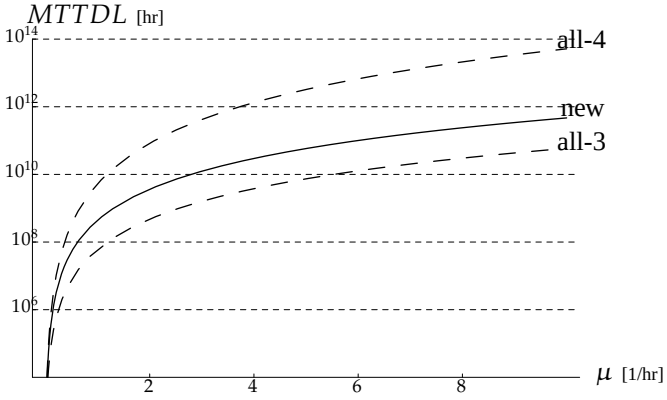

Figure 8. $MTTDL$ curves under random failures for the new codes, for all-3-erasure and for all-4-erasure correcting codes. Under random failures, the new codes are order of magnitude better than all-3-erasure correcting codes, and two orders of magnitude inferior to all-4-erasure correcting codes.

outperform 3-random erasure codes by an order of magnitude, despite having the same encoding complexity, the same update complexity, and asymptotically the same decoding complexity. However, in the presence of pure random erasures, the new codes are still two orders of magnitude worse than 4-random erasure-correcting codes.

To compare the new codes and 4-random erasure codes in the presence of both random and clustered failures, the state diagram of the new codes in Figure 7 needs to be modified to include additional states that represent clustered failures. The state diagram of 4-random failure codes in Figure 6 remains the same since this code is agnostic to the distinction between random and clustered failures. To take clustered failures into account in the Markov failure model, we add the following assumptions to those of the previous sub-section.

- Times to clustered failures (failures that are adjacent to an unrepaired previous failure) are exponentially distributed with mean $1/\chi$. Times to random failures (failures that are not clustered) are exponentially distributed with mean $1/\Omega$. The two rates sum to the total failure rate as in the random-only case: $\Lambda = \chi + \Omega$.
- The exponentially-distributed repair process eliminates isolated failures before clustered ones.

With these assumptions, the state diagram of arrays protected with the new codes under random and clustered failures is given in Figure 9. States $2'$,$3'$ and $4'$ in the upper branch represent 2, 3 and 4 clustered (not all-isolated) failures, respectively. The transitions marked with χ represent moving

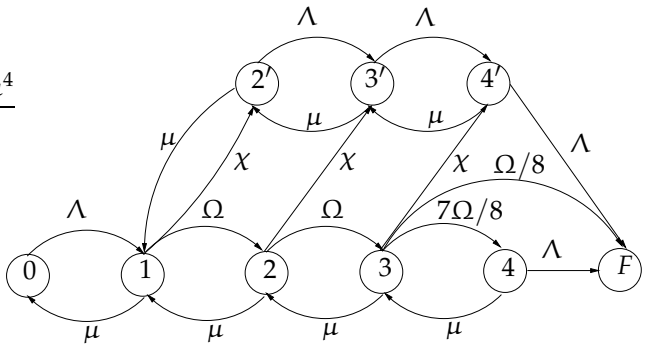

Figure 9. State diagram description of the new codes under random and clustered failures. The rate Λ failure process is now divided to a clustered-failure rate χ and random-failure rate Ω.

from all-isolated failures to a clustered failure combination. At the upper branch, both random and additional clustered failures result in clustered failure combinations – and that accounts for the transitions marked Λ, the total failure rate. From state 0, a clustered failure is undefined, hence the totality of the failure process moves to the same state 1.

Solving the 8×8 linear system for the diagram in Figure 9 (expression omitted), the $MTTDL$ can be calculated in closed form for all combinations of χ, Λ, μ. The resulting $MTTDL$ curves for the new codes under three different χ values are plotted in Figure 10, and compared to the $MTTDL$ of a 4-random failure code under the same conditions (4-random codes give the same $MTTDL$ independent of the ratio between χ and Λ). The curves of Figure 10 prove that as clustered failures become more dominant, the reliability of the new codes is approaching the reliability of a 4-random erasure-correcting code. This by itself is not a surprising result, but the ability to calculate the exact expected reliability for arbitrary χ values is a useful tool for the deployment of the new codes in real storage systems.

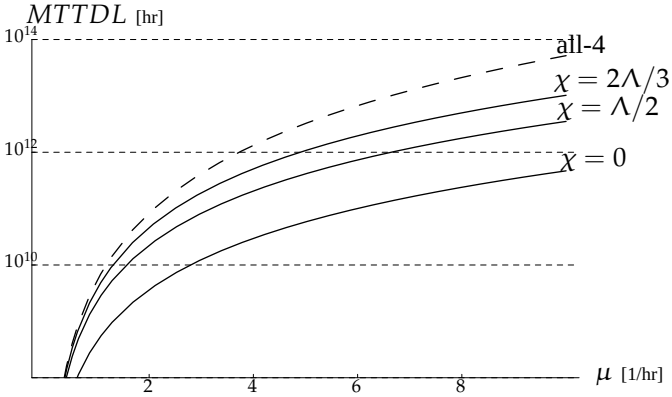

Figure 10. $MTTDL$ curves under random and clustered failures for the new codes and for all-4-erasure correcting code. For three values of χ, the $MTTDL$ of the new codes is shown by the solid curves. The $MTTDL$ of an all-4-erasure correcting code is the same for all values of χ.

VIII. Code Evaluation and Comparison with Existing Schemes

We compare the new codes to EVENODD ($r = 4$) codes using various performance criteria. For correction of 4-erasures, the EVENODD ($r = 4$) is the best known code. The comparison results are first summarized in Table I. The erasure-correction properties in Table I apply to any prime p such that 2 is primitive in GF(p).

	New codes	4-EVENODD
Code Length (up to)	$2p$	p
Redundancy	4	4
Encoding Complexity	$3kp$	$4kp$
Decoding Complexity	$3kp$	$4kp$
Update Complexity	5	7
Clustered Erasures	$\sim$ All	All
Random Erasures	$7/8$	All

TABLE I
Comparison between the new codes and EVENODD codes

The redundancy r is 4 for both codes. The new codes can support up to $2p$ information columns while EVENODD can only have up to p. Since parity columns $R0$ and $S1$ each depend on half of the information columns, the encoding complexity of the new codes is $3kp$, compared to $4kp$ in EVENODD. In both cases, when k is of the same order of p, the decoding complexity is dominated by syndrome calculations (for the new codes this has been shown in section VI). Therefore, similarly to the encoding case, the new codes need about $3kp$ bit operations to decode, compared to $4kp$ for EVENODD. As for the update-complexity, the new codes are significantly more efficient. Their small-write update complexity is 5. Each of the $2p(p-1)$ updated information bits needs 3 parity updates: $P, Q, R0$ for bits in even columns and $P, Q, S1$ for bits in odd columns. The $4(p-1)$ bits that belong to EO diagonals ($2(p-1)$ in Q and $p-1$ in each of $R0, S1$) require additional $p-1$ parity-bit updates each for adjusting even/odd parities. The small-write update-complexity of the new code is then obtained by averaging

$$\frac{6p(p-1)+4(p-1)^2}{2p(p-1)} = 5 - o(1)$$

Recall that EVENODD has small-write update-complexity of $2r - 1 - o(1) = 7 - o(1)$. The full-column update-complexity of the new code is 3 while EVENODD's is 4. Thus the new code offers a 28.57% improvement in the average number of small-writes and 25% improvement in the number of full-column updates. The fraction of clustered erasures correctable by the new codes is $1 - o(1)$, essentially the same as EVENODD's. Only in random erasure-correction capability are the new codes inferior to EVENODD codes: the fraction of correctable random erasures is $7/8 - o(1)$ compared to 1 for EVENODD.

Appendix A
Array codes: introductory example

The idea behind array codes is that the code is defined on two-dimensional arrays of bits (or groups of bits), and encoding and decoding operations are performed over the binary alphabet, using simple eXclusive OR operations. An example of an array code with two parity columns that can recover from any two column erasures is given below. The $+$ signs represent binary eXclusive OR operations. The three left columns contain pure information and the two right columns contain parity bits that are computed from the information bits as specified in the chart below.

a	b	c	$a+b+c$	$a+f+e+c$
d	e	f	$d+e+f$	$d+b+e+c$

Like encoding, decoding is also performed using simple eXclusive OR operations. For example, recovering the bits a, b, d, e at the two leftmost columns is done by the following chain of computations.

$$\begin{aligned} e &= c + (a+b+c) + (d+e+f) + (a+f+e+c) \\ &\quad + (d+b+e+c) \\ d &= e + f + (d+e+f) \\ a &= c + f + e + (a+f+e+c) \\ b &= c + a + (a+b+c) \end{aligned}$$

It is left as an exercise to verify that any two column erasures can be recovered by the code above. The small-write update complexity (the qualifier *small-write* is often omitted) of an array code is the number of parity-bit updates required for a single information-bit update, averaged over all the array information bits. In the sample code above, each of the bits a, b, d, f requires 2 parity-bit updates, and each of e, c requires 3 parity-bit updates. The update complexity of that sample code is hence $(4 \cdot 2 + 2 \cdot 3)/6 = 2.333$.

References

[1] A. Aho, J. Hopcroft, and J. Ullman, *The design and analysis of computer algorithms*. Reading, MA USA: Addison-Wesley, 1974.

[2] L. Bahl and R. Chien, "Multiple-burst-error correction by threshold decoding," *Information and Control*, vol. 15, no. 5, pp. 397–406, 1969.

[3] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: an efficient scheme for tolerating double disk failures in RAID architectures," *IEEE Transactions on Computers*, vol. 44, no. 2, pp. 192–202, 1995.

[4] M. Blaum, J. Bruck, and A. Vardy, "MDS array codes with independent parity symbols," *IEEE Transactions on Information Theory*, vol. 42, no. 2, pp. 529–542, 1996.

[5] M. Blaum and R. Roth, "New array codes for multiple phased burst correction," *IEEE Transactions on Information Theory*, vol. 39, no. 1, pp. 66–77, 1993.

[6] G. Gibson, *Redundant Disk Arrays*. Cambridge MA, USA: MIT Press, 1992.

[7] C. Huang, M. Chen, and J. Li, "Pyramid codes: flexible schemes to trade space for access efficiency in reliable data storage systems," in *Proceedings of the Sixth IEEE International Symposium on Network Computing and Applications*, Cambridge, MA USA, 2007.

[8] C. Huang and L. Xu, "STAR: An efficient coding scheme for correcting triple storage node failures," in *Proceedings of the 4th USENIX Conference on File and Storage Technologies*, San-Francisco CA, 2005.

[9] S. Lin and D. Costello, *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, Englewood Cliffs, NJ, 1983.

[10] F. MacWilliams and N. Sloane, *The Theory of Error-Correcting Codes*. Amsterdam, The Netherlands: North Holland, 1977.

[11] J. Paris and A. Amer, "Using shared parity disks to improve the reliability of RAID arrays," in *Proc. of the 28th International Performance of Computers and Communication Conference (IPCCC 2009)*, Phoenix, AZ, Dec. 2009.

[12] J. Paris and D. Long, “Using device diversity to protect data against batch-correlated disk failures,” in *Proc. of the 2nd International Workshop on Storage Security and Survivability (StorageSS 2006)*, Alexandria, VA, Oct. 2006.
[13] D. A. Patterson, G. A. Gibson, and R. Katz, “A case for redundant arrays of inexpensive disks,” in *Proc. SIGMOD Int. Conf. Data Management*, 1988, pp. 109–116.